WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Thursday, September 23, 2004

Hide?	Set Name	Query	Hit Count
		DB=USPT; PLUR=YES; OP=ADJ	
☐	L9	L8 and l2	0
☐	L8	element near4 collection near5 vector	23
☐	L7	element near4 collection near5 vector near5 call	0
☐	L6	L5 and l4	2
☐	L5	L3 and collection	2
☐	L4	l2 and enumeration	3
☐	L3	l2 and wrapper	2
☐	L2	20010329	13
☐	L1	J2EE	22

END OF SEARCH HISTORY

Previous Doc Next Doc Go to Doc#
First Hit

Generate Collection

L10: Entry 1 of 9 File: PGPB May 24, 2001

DOCUMENT-IDENTIFIER: US 20010001881 A1
TITLE: Methods and media for utilizing symbolic expressions in circuit modules

Application Filing Date:
20001219

Detail Description Paragraph:
[0129] An "enumeration" is a Java language construct that provides a uniform interface for iterating (stepping one by one through) all the elements in a collection of objects. The following three methods return enumerations of nets, ports and child SIMs of the subject SIM.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L10: Entry 4 of 9 File: USPT Sep 24, 2002

DOCUMENT-IDENTIFIER: US 6457164 B1
TITLE: Hetergeneous method for determining module placement in FPGAs

Application Filing Date (1):
20000629

Detailed Description Text (24):
An "enumeration" is a Java language construct that provides a uniform interface for iterating (stepping one by one through) all the elements in a collection of objects. The following three methods return enumerations of nets, ports and child SIMs of the subject SIM. Enumeration netElements( ):Returns an enumeration of the nets of the subject SIM. Enumeration portElements( ):Returns an enumeration of the ports of the subject SIM. Enumeration simElements( ):Returns an enumeration of the child SIMs of the subject SIM.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Search Forms
Search Results
Help
User Searches
Preferences
Logout

Generate Collection

L10: Entry 9 of 9 File: USPT Apr 10, 2001

DOCUMENT-IDENTIFIER: US 6216258 B1
** See image for Certificate of Correction **
TITLE: FPGA modules parameterized by expressions

Application Filing Date (1):
19980327

Detailed Description Text (65):
An "enumeration" is a Java language construct that provides a uniform interface for iterating (stepping one by one through) all the elements in a collection of objects. The following three methods return enumerations of nets, ports and child SIMs of the subject SIM.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L2: Entry 4 of 13 File: USPT Jun 1, 2004

DOCUMENT-IDENTIFIER: US 6745387 B1
TITLE: Method for using a transaction service synchronization interface to perform internal state clean up

Application Filing Date (1):
19990614

Drawing Description Text (4):
FIG. 2 is a diagram of a typical J2EE reference implementation; and

Detailed Description Text (3):
Sun Microsystem's Java 2 Platform Enterprise Edition (J2EE) is a platform for constructing Java technology based multi-tier applications across a distributed, object-oriented enterprise. One specific J2EE-compliant implementation is known as the J2EE Reference Implementation or J2EE RI. The J2EE RI includes an implementation of a Java Transaction Service (JTS) and a Java Transaction API (JTA). The JTS is specified by the Java Transaction Service draft specification, version 0.95, Mar. 1, 1999, and the JTA is specified by the Java Transaction API specification, version 1.0.1, Apr. 29, 1999, both specifications herein incorporated by reference. The JTA specification specifies the high-level interfaces between a Transaction Manager and any other parties involved in a distributed transaction system, such as the application, the resource manager, and the application server. The JTS specification specifies the Transaction Manager, which supports the JTA specification at the high-level and implements the OTS 1.1 Specification at the low-level. The present invention can be implemented using a Transaction Manager compliant with either OTS or JTA, or both OTS and JTA.

Detailed Description Text (4):
A typical J2EE implementation 10 is illustrated in FIG. 2. The J2EE implementation 10 may be installed on one or more physical machines, as the J2EE standard does not specify any predefined hardware configuration. The standard architecture supports two types of components, Web components and Enterprise Java Bean (EJB) 141, 142 components, and also application clients. The Web components include Java Server Pages (JSP) 121 and servlets 122. The JSP components 121 may be accessed, for example, by a web browser 20 using HTTP. Each type of component is contained within a "Container" 12, 14, 16, 17 which provides an environment within which components or applications run. A J2EE implementation 10 may contain multiples of the same type of containers 16, 17 and the containers may exist in a distributed environment. Individual components can interact with each other across containers or with external databases 18. Each container has its own Transaction Manager 124, 144, 164 to manage transactions within the container. The Transaction Managers can communicate with other Transaction Managers to determine transaction status information.

Detailed Description Text (5):
J2EE RI provides a distributed application server environment and keeps track of various states for each component in each transaction. In general, only the Transaction Managers 124, 144, 164, however, know when a transaction has completed. For example, the J2EE RI might get involved in a transaction that is started by a

client on a remote machine, but only the remote client and the Transaction Managers 124, 144, 164 know when the a transaction has completed. This creates a system maintenance problem for a J2EE RI environment. Specifically, without knowing when a transaction has completed, the J2EE RI does not know when to perform transient state clean up within each container 12, 14, 16, 17. Each container 12, 14, 16, 17 has memory space 123, 143, 163 used to store internal state information for each transaction. If the memory space 123, 143, 163 is not freed after a transaction has completed, the amount of space available for other transactions is reduced (causing "memory leaks"). Over time, these memory leaks may reduce system performance or even cause the system to crash. One possible solution is to modify the Transaction Manager implementation to perform the state clean up, but this would result in an undesirable coupling between the Transaction Manager and the J2EE RI. Also, any changes to the Transaction Manager interface may make it non-compliant with OTS 1.1. Thus, it would be desirable to implement an internal state clean up mechanism for each container after a transaction has completed, without modifying the current Transaction Manager implementation.

Detailed Description Text (6):
As described above, the J2EE system 10 needs to clean up the internal states in each container 12, 14, 16, 17 after a transaction completes, so that the allocated memory 123, 143, 163 is not used up. However, since one transaction may span multiple machines and processes, it is difficult for all the containers to know when to perform a clean up operation. The present invention uses the Synchronization Interface of a Transaction Manager that supports OTS or JTA in a unique way to trigger each container involved in a transaction to perform a clean up operation. The present mechanism has not previously been proposed or recommended, but may be implemented without adversely affecting the Synchronization Interface operation.

Detailed Description Text (7):
The procedure of the present invention is illustrated in FIG. 3. According to the present invention, the J2EE RI registers a Synchronization object with the Transaction Manager for each transaction that is started, or when the J2EE RI first learns it is involved with a transaction (step 40). When a transaction completes (step 42), the Transaction Manager invokes a method (operation) on the Synchronization object (step 44). Upon the invocation of this method, the J2EE RI performs the necessary container state clean up for this transaction (step 46). Each Transaction Manager knows when a transaction is being completed, since it is in communication with all other Transaction Managers. Thus, via a Synchronization object, distributed state clean up in the containers can be performed, without adding additional code or complexity.

CLAIMS:

4. The method of claim 3, wherein the computer system is a Java 2 Platform Enterprise Edition (J2EE) compliant system.

10. A computer readable medium as recited in claim 8, wherein the computer system is a Java 2 Platform Enterprise Edition (J2EE) compliant system.

14. A container-based distributed computer system as recited in claim 12, wherein the container-based distributed computer system is a Java 2 Platform Enterprise Edition (J2EE) compliant system.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L2: Entry 3 of 13 File: USPT Jun 22, 2004

DOCUMENT-IDENTIFIER: US 6753889 B1
TITLE: Platform independent business to business messenger adapter generation tool

Application Filing Date (1):
20001030

Brief Summary Text (7):
One approach to solving the problems of cross platform communication is to use component based, multi-tier applications based on, for example, Java 2 Enterprise Edition (J2EE) technology from Sun Microsystems Inc. of Mountain View, Calif. J2EE technology, in the form of an J2EE server, represents a multi-tier design that simplifies developing, deploying, and maintaining enterprise applications. It enables developers to focus on the specifics of programming their business logic, relying on the J2EE server to provide system services, and client-side applications (both stand alone and within web browsers) to provide the user interaction. Once developed, business logic can be deployed on servers appropriate to existing needs of an organization.

Brief Summary Text (8):
Although J2EE server technology substantially solves many of the problems related to cross platform performance, there still remains the need to provide a mechanism whereby an e-business and its respective e-business partners can reliably conduct an e-business transaction.

Drawing Description Text (4):
FIG. 2 illustrates a J2EE based implementation of the enterprise computer based e-business in accordance with an embodiment of the invention.

Drawing Description Text (8):
FIG. 6 illustrates a situation where the partner system is a J2EE based enterprise computer system.

Detailed Description Text (3):
In general, a reliable and easy to use business to business (B2B) message adapter generation tool for use in describing a B2B message adapter in an enterprise computer system is described. In one embodiment, the enterprise computer system is a J2EE based enterprise computer system. The B2B messenger is coupled to a Java Message Service API (referred to as JMS) that provides an interface between the B2B messenger and the various business components included in the J2EE based enterprise computer system. In the described embodiment, the B2B messenger subscribes to a Java Messenger Server (JMS) topic based upon an associated subscription rule. By subscribes, it is meant that the B2B messenger "listens" to a particular JMS topic that is identified with a particular subscription rule. When the JMS topic points to a particular native message (referred to as a JMS message), a subscription manager included in the messenger receives the JMS message and directs a message adapter to modify the JMS message into a format consistent with a receiving partner based upon both the corresponding subscription rule and a corresponding document template, or B2B schema. The message adapter is provided by the inventive B2B message adapter generation tool that takes as its two input classes, a sender

message format and a receiver message format which are then heuristically analyzed to form a correspondence mapping between the two formats. In a preferred embodiment, the B2B schema is used as a template to assure that the document sent to the receiving partner conforms to the partner's documentation rules.

Detailed Description Text (5):
Once received, the receiver partner returns a response message by way of a receiver partner adapter, which typically takes the form of a servlet. As directed by a delivery manager using both a delivery rule and the B2B schema, a receiver message adapter converts the response to a received JMS message, which is then delivered to a corresponding JMS topic in the JMS. In this way, the e-business is able to communicate in a loosely coupled manner with the associated partner without the requirement of knowing what form the partner's portion of the B2B contract takes, and vice versa. By loosely coupling the two portions of the B2B contract, the inventive messenger provides for B2B integration between J2EE based businesses and its partners, that may or may not be J2EE based.

Detailed Description Text (8):
Although, the invention will initially be described in terms of an e-business messenger as part of a J2EE based enterprise computer system, the present invention can be used in any networked computer system that uses JMS as its messaging infrastructure.

Detailed Description Text (9):
With reference to FIG. 1, a Business to Business to Customer (B2B2C) system 100 in accordance with an embodiment of the invention is shown. The system 100 includes an inventive Java 2 Enterprise Edition (J2EE.TM.) based enterprise computer system type e-business 102 capable of reliably (and asynchronously) communicating with any number of associated partners regardless of their respective transport protocols, document schemas, etc. In the described embodiment, the e-business 102 is coupled to an e-customer 104 (to form the B2C portion of the system 100) and a variety of independent buyers 106, suppliers 108, as well as an e-market 110, each of which can, and usually does, have its own standards and practices for conducting a business transaction. It should be noted that the e-market 110 and any of the partner systems can be in any technology other than J2EE.

Detailed Description Text (19):
With reference to FIG. 3 it is well established that one of the basic building blocks of any J2EE based enterprise computer system is what is referred to as an J2EE server. FIG. 3 illustrates the architecture of a J2EE server 300 in accordance with an embodiment of the invention. In the described embodiment, the J2EE server 300 is a collection of services for supporting an EJB installation. These services include management of distributed transactions, management of distributed objects and distributed invocations on these objects, and low-level system services. In short, the J2EE server 300 manages the resources needed to support an EJB component (or Bean) 302 that is included in an EJB container 304. In the described embodiment, the EJB container 304 is a home for EJB components such as EJB component 302 providing a scalable, secure, transactional environment in which EJB components can operate. The EJB container 304 handles the object life cycle, including creating and destroying an object as well as handling the state management of EJB components.

Detailed Description Text (21):
When a Bean 302 is installed on the J2EE server 300, a remote interface referred to as an EJB Object 310 is automatically generated. The EJB Object 310 is an object that exposes only the remote interface specified by the programmer. In this way, the EJB Object 310 acts like a proxy, intercepting any remote object invocations and calling the appropriate methods on the enterprise Bean instance. The EJB container 304 implements the EJB Home interface 308 of each Bean 302 installed in the container. It allows for the creation of a Bean, deletion of a Bean and

querying information or "metadata" about a Bean.

Detailed Description Text (29):
FIG. 6 shows a situation where a first B2B messenger 602 (such as e-business 102) is in direct communication with a second B2B messenger 604 (such as, for example, e-market 110), such as one belonging to a partner system that was also built using J2EE, in accordance with an embodiment of the invention. When the first B2B messenger 602 is sending a message to the second B2B messenger 604, a sending EJB partner adapter 606 is directly coupled to a receiving servlet partner adapter 608. The second B2B messenger 604 sends the response by coupling a sending EJB partner adapter 610 directly to a receiving servlet partner adapter 612.

CLAIMS:

9. A method as recited in claim 8, wherein the enterprise computer system is a Java 2 Enterprise Edition (J2EE) based enterprise computer system.

18. A computer program product claim as recited in claim 17, wherein the enterprise computer system is a Java 2 Enterprise Edition (J2EE) based enterprise computer system.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L2: Entry 5 of 13 File: USPT Apr 13, 2004

DOCUMENT-IDENTIFIER: US 6721777 B1
TITLE: Modular and portable deployment of a resource adapter in an application server

Application Filing Date (1):
20000524

Detailed Description Text (2):
The present invention specifies requirements for packaging and deploying of a resource adapter. These requirements support a modular and portable deployment of a resource adapter into a Java 2 Enterprise Edition (J2EE) compliant application server.

Detailed Description Text (6):
A resource adapter module 206 corresponds to a J2EE module in terms of the J2EE composition hierarchy. A J2EE module represents the basic unit of composition of a J2EE application. Examples of J2EE modules include: EJB module, application client module, web client module. A resource adapter module 206 can be deployed either directly into an application server 208 as a stand alone unit, or with a J2EE application that consists of one or more J2EE modules in addition to a resource adapter module 206. The J2EE specification specifies requirements for the assembly and packaging of J2EE applications. The stand alone deployment of a resource adapter module 206 into an application server 208 is typically done to support scenarios in which multiple J2EE applications share a single resource adapter module. However in certain scenarios, a resource adapter module 206 will only be required by components within a single J2EE application. The deployment option of resource adapter module 206 bundled with a J2EE application supports the latter scenario. During the deployment, the deployer installs a resource adapter module 206 in an application server 208 and then configures it in the target operational environment.

Detailed Description Text (60):
The eXtensible Markup Language (XML) Document Type Definition (DTD) may be used for the deployment descriptor for resource adapter 202. The comments in the DTD specify additional requirements for the syntax and semantics that cannot be specified by the DTD mechanism. All valid resource adapter deployment descriptors contain the following DOCTYPE declaration: <!DOCTYPE connector PUBLIC "//Sun Microsystems, Inc.//DTD Connector 1.0/ /EN" "http://java.sun.com/j2ee/dtds/connector_1_0.dtd">

Other Reference Publication (2):
Bill Shannon, "Java 2 Platform Enterprise Education Specification Version 1.2", Dec. 17, 1999, Internet: <URL:http://java.sun.com/j2ee/docs.html>.

Other Reference Publication (3):
D. Coward et al, "Java 2 Platform, Enterprise Edition (J2EE) Reference Implementation", 1999, Internet:
<URL:http://servlet.java.sun.com/javaone/javaone00/pdfs/e675.pdf>.

CLAIMS:

1. A computer system having a client server architecture, comprising: an application server; an enterprise information system including computing resources; a connection provider; a resource adapter, said resource adapter implemented by said connection provider to serve said resources from said enterprise information system to clients, wherein the resource adapter is packaged with a deployment descriptor to form a resource adapter module corresponding to a Java 2 Enterprise Edition (J2EE) module hierarchy, wherein said resource adapter module is deployed into the application server to allow multiple J2EE applications to share the resource adapter module; and a deployer, said deployer using a deployment tool to configure the resource adapter into a target operational environment, the deployment tool capable of reading the deployment descriptor.

17. A computer program product, which, when executed by a computer, implements a client server computer system by performing the steps of: providing an application server; providing an enterprise information system including computing resources; providing a connection provider; providing a resource adapter, said resource adapter implemented by said connection provider to serve said resources from said enterprise information system to clients, wherein the resource adapter is packaged with a deployment descriptor to form a resource adapter module corresponding to a Java 2 Enterprise Edition (J2EE) module hierarchy, wherein said resource adapter module is deployed into the application server to allow multiple J2EE applications to share the resource adapter module; and providing a deployer, said deployer using a deployment tool to configure the resource adapter into a target operational environment, the deployment tool capable of reading the deployment descriptor.

18. A computer system having a client server architecture, comprising: an application server; an enterprise information system including computing resources; a connection provider; a resource adapter, said resource adapter implemented by said connection provider to serve said resources from said enterprise information system to clients, wherein the resource adapter is packaged with a deployment descriptor to form a resource adapter module corresponding to a J2EE module hierarchy, wherein said resource adapter module is deployed into the application server to allow multiple J2EE applications to share the resource adapter module, the resource adapter module specifying a level of transaction support provided by the resource adapter, the level of transaction support indicating that resource manager local transactions are supported, the resource manager local transactions being supported by implementing a LocalTransaction interface; and a deployer, said deployer using a deployment tool to configure the resource adapter into a target operational environment, the deployment tool capable of reading the deployment descriptor.

19. A computer system having a client server architecture, comprising: an application server; an enterprise information system including computing resources; a connection provider; a resource adapter, said resource adapter implemented by said connection provider to serve said resources from said enterprise information system to clients, wherein the resource adapter is packaged with a deployment descriptor to form a resource adapter module corresponding to a J2EE module hierarchy, wherein said resource adapter module is deployed into the application server to allow multiple J2EE applications to share the resource adapter module; and a deployer, said deployer using a deployment tool to configure the resource adapter into a target operational environment, the deployment tool capable of reading the deployment descriptor, wherein the deployment descriptor defines a contract between the connector provider and the deployer.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L2: Entry 13 of 13 File: USPT Jun 25, 2002

DOCUMENT-IDENTIFIER: US 6411956 B1
TITLE: Method for distributed transaction support using JDBC 1.0 drivers

Application Filing Date (1):
19990614

Brief Summary Text (16):
FIG. 3 is a diagram of a typical J2EE RI implementation;

Detailed Description Text (3):
Sun Microsystem's Java 2 Platform Enterprise Edition (J2EE) is a platform for constructing Java technology based multi-tier applications across a distributed, object-oriented enterprise. One specific J2EE-compliant implementation is known as the J2EE Reference Implementation or J2EE RI. The J2EE RI includes an implementation of a Java Transaction Service (JTS) and a Java Transaction API (JTA). The JTS is specified by the Java Transaction Service draft specification, version 0.95, Mar. 1, 1999, and the JTA is specified by the Java Transaction API specification, version 1.0.1, Apr. 29, 1999, both specifications herein incorporated by reference. The JTA specification specifies the high-level interfaces between a Transaction Manager and any other parties involved in a distributed transaction system, such as the application, the resource manager, and the application server.

Detailed Description Text (4):
A typical J2EE implementation 10 is illustrated in FIG. 3. The J2EE implementation 10 may be installed on one or more physical machines, as the J2EE standard does not specify any predefined hardware configuration. The standard architecture supports two types of components, Web components and Enterprise Java Bean (EJB) 141, 142 components, and also application clients. The Web components include Java Server Pages (JSP) 121 and servlets 122. The JSP components 121 may be accessed, for example, by a web browser 20 using HTTP. Each type of component is contained within a "Container" 12, 14, 16, 17 which provides an environment within which components or applications run. A J2EE implementation 10 may contain multiples of the same type of containers 16, 17 and the containers may exist in a distributed environment. Individual components can interact with each other across containers or with external databases 18. Each container has its own Transaction Manager 124, 144, 164 to manage transactions within the container. The Transaction Managers can communicate with other Transaction Managers to determine transaction status information.

CLAIMS:

9. The method of claim 8, wherein the computer system is a Java 2 Platform Enterprise Edition (J2EE) system.

15. The method of claim 14, wherein the computer system is a Java 2 Platform Enterprise Edition (J2EE) system.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L3: Entry 1 of 2 File: USPT Apr 13, 2004

DOCUMENT-IDENTIFIER: US 6721747 B2
TITLE: Method and apparatus for an information server

Application Filing Date (1):
20010112

Detailed Description Text (7):
The system is predominantly based on object-oriented programming principles as described in "Object-Oriented Software Construction" by Bertrand Meyer, Prentiss-Hall, 1988, ISBN 0-13-629049-3 and the Sun Microsystems.TM. developed JAVA.TM. systems described in the following publications: Enterprise JavaBeans Specification, v1.1 (can be found at //java.sun.com/products/ejb/docs.html) Enterprise JavaBeans, Richard Monson-Haefel, O'Reilly. Enterprise JavaBeans: Developing Component-Based Distributed Applications, Tom Valesky, Addison-Wesley. Enterprise JavaBeans Developer's Guide (Beta Version) at //developer.java.sun.com/developer/earlyAccess/j2sdkee/doc-beta/guides/ejb/html/TOC.html J2EE Application Programming Model (Beta Release), at //developerjava.sun.com/ developer/earlyAccess/j2sdkee/download-docs.html

Detailed Description Text (226):
for Boolean, Timestamp, String, Integer, Float, and Double data types. The ISabaHome interface provides a layer of abstraction over the standard EJB interface EJBHome. The BDK also defines a class SabaPrimaryKey (a thin wrapper around the String class) which can be used by entity beans for defining primary keys.

Detailed Description Text (243):
In the alternative embodiment, the Business Server embodies a development kit framework which provides a set of interfaces and classes in the form of Java packages, identifies certain services that developers can rely on, and defines an application development model. The framework relies extensively on the server-side component model espoused by Java, namely Enterprise JavaBeans (EJB) components. Selection of EJBs as the server-side component model is driven in part by the requirements of reliance on open standards and backward compatibility. Using EJBs also enables integration with other Java 2 Enterprise Edition (J2EE) technologies such as Java ServerPages (JSP) and servlets that one would intend to use for web applications development. Furthermore, a number of EJB-enabled application servers available in the marketplace could be used to deploy the components so developed.

Detailed Description Text (247):
1. Provide an additional layer of abstraction (by writing wrappers around base Java classes) to provide a richer level of functionality needed by SABA applications and to allow future modifications with minimal impact on the client application code.

Detailed Description Text (256):
The classes Class and Field by themselves, however, may not provide the rich functionality needed by certain applications. For instance, there is no way to indicate minimum and maximum values of an attribute in the Field class. Thus, what is needed is to create new classes that provide wrappers around Class and Field and capture the additional information. In the interest of consistency with previously

used names while avoiding conflicts at the same time, two new classes may be used: SabaPlatformClass (inherits from Class) and SabaPlatformattribute (inherits from Field). In addition to the functionality provided by Class (e.g., for getting parent class), SabaPlatformClass provides for such additional functionality as domain-based attributes and getting fixed vs. extended custom attribute counts. Similarly, SabaPlatformattribute provides functionality for LOVs, default value, and minimum and maximum values. (As we will discuss later, the classes SabaPlatformClass and SabaPlatformAttribute themselves are beans--or, entity beans to be more specific--in this alternative embodiment system.)

Detailed Description Text (270):
for Boolean, Timestamp, String, Integer, Float, and Double data types. The ISabaHome interface provides a layer of abstraction over the standard EJB interface EJBHome. The BDK also defines a class SabaPrimaryKey (a thin wrapper around the String class) which can be used by entity beans for defining primary keys.

Detailed Description Text (278):
Vendor-Specific Wrappers

Detailed Description Text (279):
In the alternative embodiment, when some areas within the J2EE specifications are still not standardized and are left up to individual vendors for implementation, additional facilities will be needed. To prevent vendor-specific implementation details from migrating into SABA code, the BDK would provide a class SabaJ2EEVendor that provides a wrapper around vendor-specific implementations. SabaJ2EEVendor provides static methods that can be used to perform activities in a vendor-neutral fashion in SABA code. An example method in SabaJ2EEVendor is getInitialContext ( ), which encapsulates the logic for getting an initial context (at present, the mechanism for this is vendor-dependent). To use a particular vendor's implementation of J2EE specifications, one will have to provide implementations of the methods in this class. By default, the BDK will provide implementations of this class for a few selected J2EE servers.

Detailed Description Text (333):
In the alternative embodiment, every class in an application does not have to be a bean. Indeed, with the overhead of locating a bean through a naming service and going through the home and remote interfaces of a bean to perform useful work would negatively impact performance (though some servers will optimize the process for beans located within the same virtual machine). The application developers can implement selected classes as helper classes and not as beans. Sun Microsystems' J2EE Application Programming Model identifies certain instances where helper classes are applicable. One such example is dependent classes that can only be accessed indirectly through other classes (beans). Sun's J2EE APM offers CreditCard and Address classes as examples of a dependent classes.

Detailed Description Text (334):
EJBs are packaged as EJB jar files that are comprised of the class files for the bean class, the home interface, the remote interface, the primary key class (if applicable), in addition to the deployment descriptor and a manifest. The jar file can be created using the jar application supplied with JDK, or by using some GUI front-end utility provided by the J2EE server being used. The deployment mechanism varies with the servers. For Weblogic server, an entry can be made in the weblogic.properties file; for Sun's reference implementation, the deploytool utility can be used to achieve this in an interactive manner.

Detailed Description Text (620):
Section 1 defines the namespaces used in the stylesheet. Section 2 defines the root level template. This template produces the html tags, and generates the html head element complete with the title element. Section 3 defines the default template: every element, attribute, text and comment is copied to the resulting document,

unless a more specific template provides different instructions. Section 4 specifies a template for eliminating the wdk:head and wdk:widgets elements and their contents (since the contents of these tags should not be rendered using the default template defined in section 3). Section 5 introduces a template for transforming every widget by wrapping them into a span element replacing the wdk:widget "wrapper". This makes it possible to use CSS styling on a per named-widget basis. Finally, section 6 defines the template for processing the wdk:page element.

Detailed Description Text (742):
Referring to FIG. 9, these Connectors services may include Monitor 945, Accessor 935, Importer 940, and Updater (not shown). Accessors, Importers, and Updaters are essentially thin wrappers around XSL stylesheet operations. They translate documents between native formats and the Interchange format using a predefined stylesheet. These connector services may also contain additional logic for cases where a single Interchange format document represents multiple native documents, and vice versa. A more detailed description of the service components for these Connector services and their implementation on the Interconnect Backbone follows.

Detailed Description Text (791):
The rationale behind this separation is to allow for the Interconnect DeliveryService/PMM to be deployed across a wide variety of communication protocols. Supporting a new protocol requires building a delivery transport that wraps that protocol. The protocol wrappers are implemented as peers, and initiate and accept connections, send and receive messages, terminate gracefully, etc. For example, the following steps would be performed to build a TCP/IP socket Interconnect Transport:

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L6: Entry 1 of 2 File: USPT Apr 13, 2004

DOCUMENT-IDENTIFIER: US 6721747 B2
TITLE: Method and apparatus for an information server

Application Filing Date (1):
20010112

Detailed Description Text (2):
The present invention provides a solution to the needs described above through a system and method for integrating the disparate applications, and managing the applications processes in a hardware resource and user effort efficient manner. The automated system of the present invention uses a business systems platform architecture comprised of several unique servers in a base platform (the "Platform") to efficiently manage multiple applications which may themselves generally be distributed across a network. The platform makes use of a collection of Core Services which provide additional security, internationalization services, and reporting services which are applicable to all applications. The Core Services are made available to a multitude of common business objects, which themselves are made available to various applications.

Detailed Description Text (7):
The system is predominantly based on object-oriented programming principles as described in "Object-Oriented Software Construction" by Bertrand Meyer, Prentiss-Hall, 1988, ISBN 0-13-629049-3 and the Sun Microsystems.TM. developed JAVA.TM. systems described in the following publications: Enterprise JavaBeans Specification, v1.1 (can be found at //java.sun.com/products/ejb/docs.html) Enterprise JavaBeans, Richard Monson-Haefel, O'Reilly. Enterprise JavaBeans: Developing Component-Based Distributed Applications, Tom Valesky, Addison-Wesley. Enterprise JavaBeans Developer's Guide (Beta Version) at //developer.java.sun.com/developer/earlyAccess/j2sdkee/doc-beta/guides/ejb/html/TOC.html J2EE Application Programming Model (Beta Release), at //developerjava.sun.com/ developer/earlyAccess/j2sdkee/download-docs.html

Detailed Description Text (226):
for Boolean, Timestamp, String, Integer, Float, and Double data types. The ISabaHome interface provides a layer of abstraction over the standard EJB interface EJBHome. The BDK also defines a class SabaPrimaryKey (a thin wrapper around the String class) which can be used by entity beans for defining primary keys.

Detailed Description Text (243):
In the alternative embodiment, the Business Server embodies a development kit framework which provides a set of interfaces and classes in the form of Java packages, identifies certain services that developers can rely on, and defines an application development model. The framework relies extensively on the server-side component model espoused by Java, namely Enterprise JavaBeans (EJB) components. Selection of EJBs as the server-side component model is driven in part by the requirements of reliance on open standards and backward compatibility. Using EJBs also enables integration with other Java 2 Enterprise Edition (J2EE) technologies such as Java ServerPages (JSP) and servlets that one would intend to use for web applications development. Furthermore, a number of EJB-enabled application servers

available in the marketplace could be used to deploy the components so developed.

Detailed Description Text (247):
1. Provide an additional layer of abstraction (by writing wrappers around base Java classes) to provide a richer level of functionality needed by SABA applications and to allow future modifications with minimal impact on the client application code.

Detailed Description Text (256):
The classes Class and Field by themselves, however, may not provide the rich functionality needed by certain applications. For instance, there is no way to indicate minimum and maximum values of an attribute in the Field class. Thus, what is needed is to create new classes that provide wrappers around Class and Field and capture the additional information. In the interest of consistency with previously used names while avoiding conflicts at the same time, two new classes may be used: SabaPlatformClass (inherits from Class) and SabaPlatformattribute (inherits from Field). In addition to the functionality provided by Class (e.g., for getting parent class), SabaPlatformClass provides for such additional functionality as domain-based attributes and getting fixed vs. extended custom attribute counts. Similarly, SabaPlatformattribute provides functionality for LOVs, default value, and minimum and maximum values. (As we will discuss later, the classes SabaPlatformClass and SabaPlatformAttribute themselves are beans--or, entity beans to be more specific--in this alternative embodiment system.)

Detailed Description Text (270):
for Boolean, Timestamp, String, Integer, Float, and Double data types. The ISabaHome interface provides a layer of abstraction over the standard EJB interface EJBHome. The BDK also defines a class SabaPrimaryKey (a thin wrapper around the String class) which can be used by entity beans for defining primary keys.

Detailed Description Text (278):
Vendor-Specific Wrappers

Detailed Description Text (279):
In the alternative embodiment, when some areas within the J2EE specifications are still not standardized and are left up to individual vendors for implementation, additional facilities will be needed. To prevent vendor-specific implementation details from migrating into SABA code, the BDK would provide a class SabaJ2EEVendor that provides a wrapper around vendor-specific implementations. SabaJ2EEVendor provides static methods that can be used to perform activities in a vendor-neutral fashion in SABA code. An example method in SabaJ2EEVendor is getInitialContext ( ), which encapsulates the logic for getting an initial context (at present, the mechanism for this is vendor-dependent). To use a particular vendor's implementation of J2EE specifications, one will have to provide implementations of the methods in this class. By default, the BDK will provide implementations of this class for a few selected J2EE servers.

Detailed Description Text (333):
In the alternative embodiment, every class in an application does not have to be a bean. Indeed, with the overhead of locating a bean through a naming service and going through the home and remote interfaces of a bean to perform useful work would negatively impact performance (though some servers will optimize the process for beans located within the same virtual machine). The application developers can implement selected classes as helper classes and not as beans. Sun Microsystems' J2EE Application Programming Model identifies certain instances where helper classes are applicable. One such example is dependent classes that can only be accessed indirectly through other classes (beans). Sun's J2EE APM offers CreditCard and Address classes as examples of a dependent classes.

Detailed Description Text (334):
EJBs are packaged as EJB jar files that are comprised of the class files for the

bean class, the home interface, the remote interface, the primary key class (if applicable), in addition to the deployment descriptor and a manifest. The jar file can be created using the jar application supplied with JDK, or by using some GUI front-end utility provided by the J2EE server being used. The deployment mechanism varies with the servers. For Weblogic server, an entry can be made in the weblogic.properties file; for Sun's reference implementation, the deploytool utility can be used to achieve this in an interactive manner.

Detailed Description Text (465):
The widget specification 888 is a list of widgets needed by the page. These widgets include input fields of all types (textboxes, radio button collections, check box collections, dropdown lists, hyperlink buttons, etc.). Besides declaring what widgets the page needs, the specification 888 can also include how these widgets relate to the data model. For example, the page may require an edit button widget for every object it displays. The widget specification 888 can therefore indicate that the edit button is "attached to" those objects. The widget specification 888 can be very incomplete, because users (such as view developers) will typically only need the name of the widget for layout purposes. The widget library will take care of rendering the widget itself.

Detailed Description Text (620):
Section 1 defines the namespaces used in the stylesheet. Section 2 defines the root level template. This template produces the html tags, and generates the html head element complete with the title element. Section 3 defines the default template: every element, attribute, text and comment is copied to the resulting document, unless a more specific template provides different instructions. Section 4 specifies a template for eliminating the wdk:head and wdk:widgets elements and their contents (since the contents of these tags should not be rendered using the default template defined in section 3). Section 5 introduces a template for transforming every widget by wrapping them into a span element replacing the wdk:widget "wrapper". This makes it possible to use CSS styling on a per named-widget basis. Finally, section 6 defines the template for processing the wdk:page element.

Detailed Description Text (742):
Referring to FIG. 9, these Connectors services may include Monitor 945, Accessor 935, Importer 940, and Updater (not shown). Accessors, Importers, and Updaters are essentially thin wrappers around XSL stylesheet operations. They translate documents between native formats and the Interchange format using a predefined stylesheet. These connector services may also contain additional logic for cases where a single Interchange format document represents multiple native documents, and vice versa. A more detailed description of the service components for these Connector services and their implementation on the Interconnect Backbone follows.

Detailed Description Text (791):
The rationale behind this separation is to allow for the Interconnect DeliveryService/PMM to be deployed across a wide variety of communication protocols. Supporting a new protocol requires building a delivery transport that wraps that protocol. The protocol wrappers are implemented as peers, and initiate and accept connections, send and receive messages, terminate gracefully, etc. For example, the following steps would be performed to build a TCP/IP socket Interconnect Transport:

Detailed Description Text (915):
In an embodiment of the invention, RDF Query Language (RQL) is an easy-to-learn, easy-to-author language for querying collections of RDF documents. It is designed to support the full functionality required by Information Distributor.

Detailed Description Paragraph Table (14):
public interface SabaPersonHome extends ISabaHome { public SabaPersonEJB

findByPrimaryKey (SabaPrimaryKey id) throws FinderException, RMIException; public Collection findByName (String fName, String lName) throws FinderException, RMIException; public SabaPersonEJB create (String fName, String lName) throws CreateException, RMIException; }

Detailed Description Paragraph Table (17):
public void ejbRemove ( ) { /* Locate the home interface (regnHome) for the ** SabaRegistration bean (code not shown) */ Collection regns = (Collection) regnHome.findByPersonID (this.id); Iterator iter = regns.iterator ( ); while (iter.hasNext ( )) { SabaRegistrationEJB registrn = (SabaRegistrationEJB) iter.next ( ); registrn.remove ( ); } }

Detailed Description Paragraph Table (26):
public interface SabaSecurityManager extends ISabaRemote { /* methods for creating and updating security lists */ public ISecurityList createSecurityList (SecurityDetail detail); public SecurityDetail getDetail (ISecurityList theSecurityList); public void update (ISecurityList theSecurityList, SecurityDetail detail); public void remove (ISecurityList theSecurityList); /* methods for adding & removing privileges to security lists */ public void addPrivilege (ISecurityList theList, IPrivilege thePrivilege, Domain theDomain); public void removePrivilege (ISecurityList theList, IPrivilege thePrivilege, Domain theDomain); /* methods for adding & removing members from security lists */ public void addMember (ISecurityList theList, ISecurityListMember theMember); public void removeMember (ISecurityList theList, ISecurityListMember theMember); /* methods to check privileges */ public boolean isMember (ISecurityList theList, ISecurityListMember theMember); public boolean hasPrivilege (ISecurityListMember theMember, IAtomicPrivilege thePrivilege, Domain theDomain); public Collection getPrivileges (ISecurityListMember theMember, IComponent theComponent, Domain theDomain); /* standard finder */ public ISecurityList findSecurityListByKey (String id); public Collection findSecurityListByName (String name); public Collection findAllSecurityLists ( ); } /* SabaSecurityManager */

Detailed Description Paragraph Table (44):
try { /* get all stylesheets referred to by this document */ Vector resources = getResources(document, request, context); /* apply each stylesheet in turn */ Enumeration e = resources.elements( ); while (e.hasMoreElements( )) { Object resource = e.nextElement( ); this.logger.log(this, "Processing stylesheet" + resource.toString( ), Logger.DEBUG); Document stylesheet = getStylesheet(resource, request, !xsltDebugger.cacheDisabled( )); Document result = this.parser.createEmptyDocument( ); document = transformer.transform(document, null, stylesheet, resource.toString( ), result, params); if (xsltDebugger.debugStylesheet(document, resource)) { // requested debug output to browser, so done now return document; } } return document; } catch (PINotFoundException e) { return document; }

Detailed Description Paragraph Table (80):
public class MatchResultSet { /** * Set the results. * @param theResults Vector of RDFDescription objects. */ public void setResults(Vector theResults) /** * Return an Enumeration of match results. * @return Enumeration of RDFDescription objects */ public Enumeration getResults( )

Detailed Description Paragraph Table (83):
<!-- An RQL document contains a single Select element. --> <!ELEMENT rdfquery (select)> <!-- Each Select clause contains a single Condition. The "properties" attribute defines the information to return as part of the result set. Note that the URI of each matching Resource is always returned. --> <!ELEMENT select (condition)> <!ATTLIST select properties NMTOKENS #IMPLIED> <!-- A Condition can either directly contain an operation, or contain a boolean grouping operator --> <! ELEMENT condition ((operation+, property, value, condition?) .vertline. and .vertline. or .vertline. not)> <!-- Boolean grouping operators --> <!ELEMENT

and (condition, condition+)> <!-- the "applies" attribute determines whether or not the condition within a grouping operation must all apply to the same value in a Collection. --> <!ATTLIST and applies (within .vertline. across) "within"> <! ELEMENT or (condition, condition+)> <!ATTLIST or applies (within .vertline. across) "within"> <!ELEMENT not (condition)> <!-- An operation defines how to compare a property to a value -- > <!ELEMENT operation (#PCDATA)> <!-- Property identifies a specific property in an RDF file. For container objects, any children are acceptable matches, and intervening Container and Description tags are automatically navigated past. --> <!ELEMENT property (#PCDATA)> <!-- A value defines the value to which a property is compared. It is either a constant String, or a Variable whose value comes from a target RDF file. --> <!ELEMENT value (#PCDATA .vertline. variable)*> <!-- The value element can have a dt:type attribute specifying its datatype --> <!ATTLIST value dt:type NMTOKEN #IMPLIED> <!-- A variable indicates a property value obtained from a target RDF file; it contains a Property element. --> <!ELEMENT variable (property)>

Previous Doc Next Doc Go to Doc#